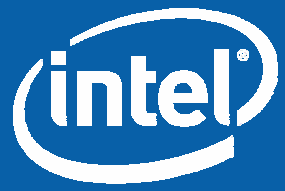


Hyper-Threading on Dual Core Intel® Itanium® 2 Processors

Rohit Bhatia
Itanium® Processor Architect
Intel® Corporation

Agenda

Multi-Threading Background

Dual Core Intel® Itanium® Hyper-Threading

- Sharing Resources
- Dedicated Resources
- A Targeted approach

Software Optimizations

- Semaphores
- Tailoring thread switch policy
- Idle loops

Hyper-Threading Is/Is Not

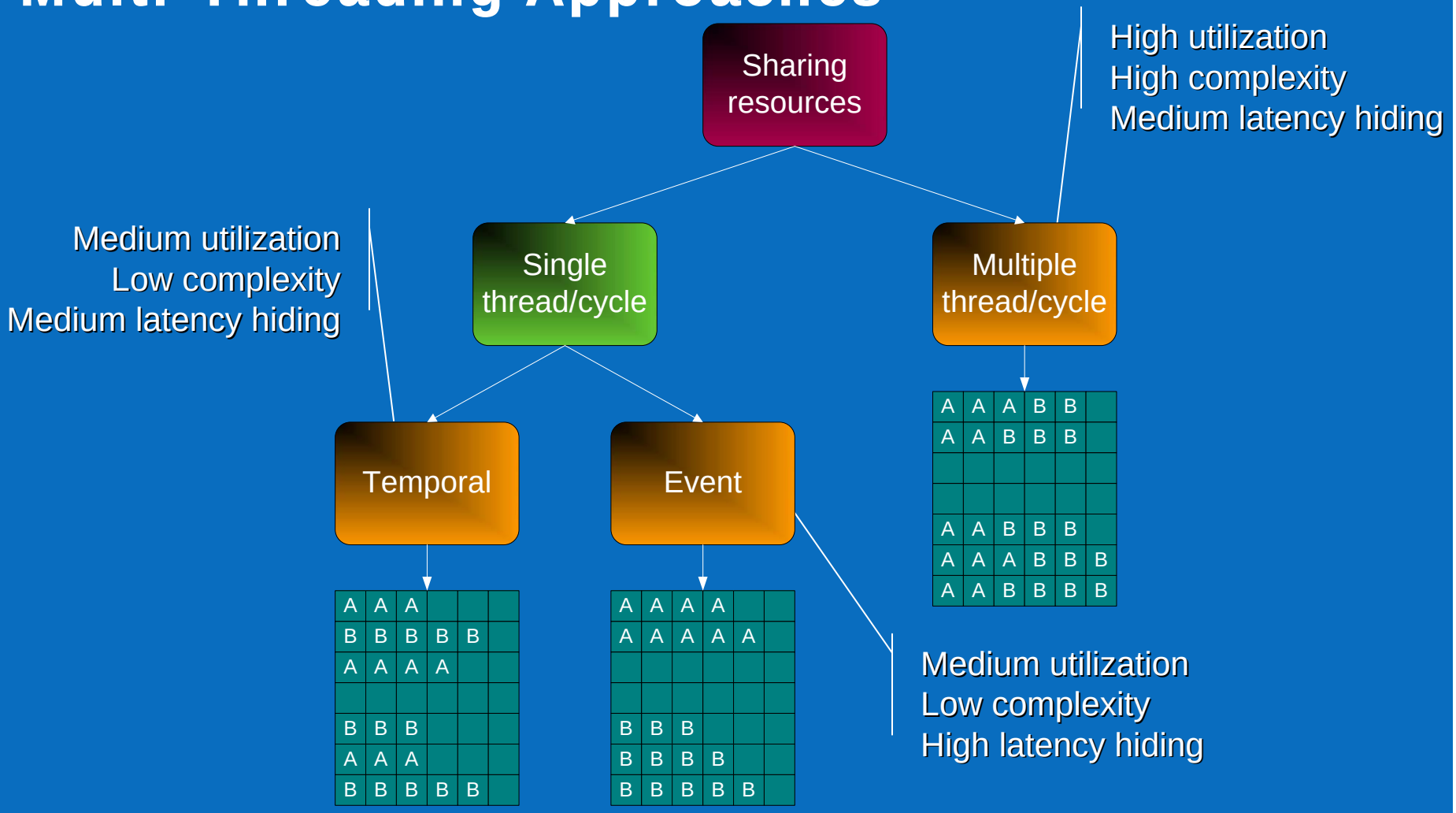
IS

- Presenting full logical processor to OS
- All application and system state is unique to each thread
- An Intel branded name for what industry and academia typically call Multi-Threading
- Supported by any multi-processor capable OS

IS NOT

- User or application level threading known to windows or pthreads
  - An user or application level threading approach is not feasible due to various user and application level threading approaches

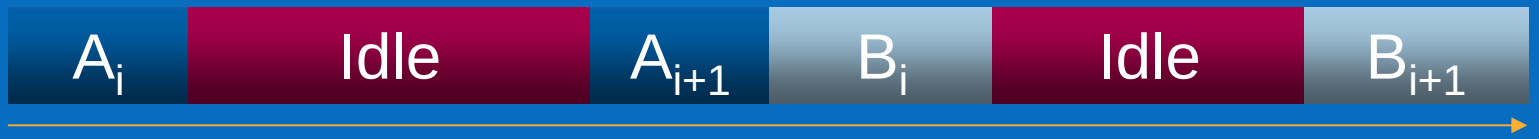
Multi-Threading Approaches



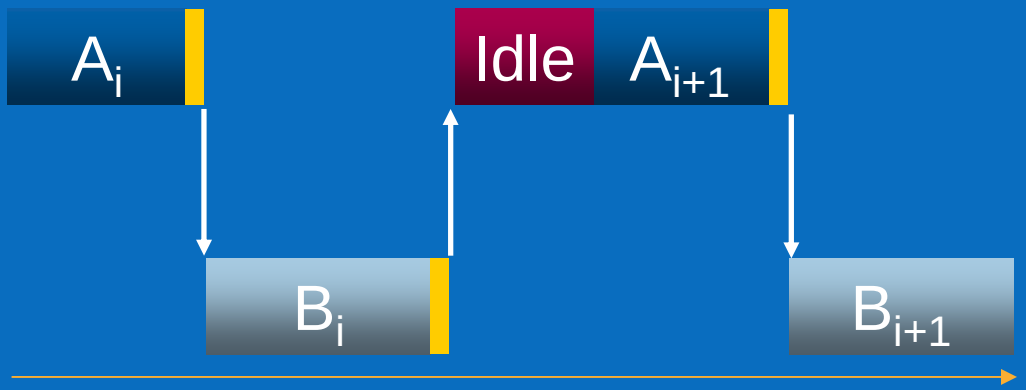
Pentium 4 uses SMT or Multiple thread/cycle approach
Dual Core Itanium® 2 uses ALL approaches

Event Multi-Threading

Serial Execution



Event Multi-Threaded Execution



Dual-Core Itanium® 2 Hyper-Threading

The core

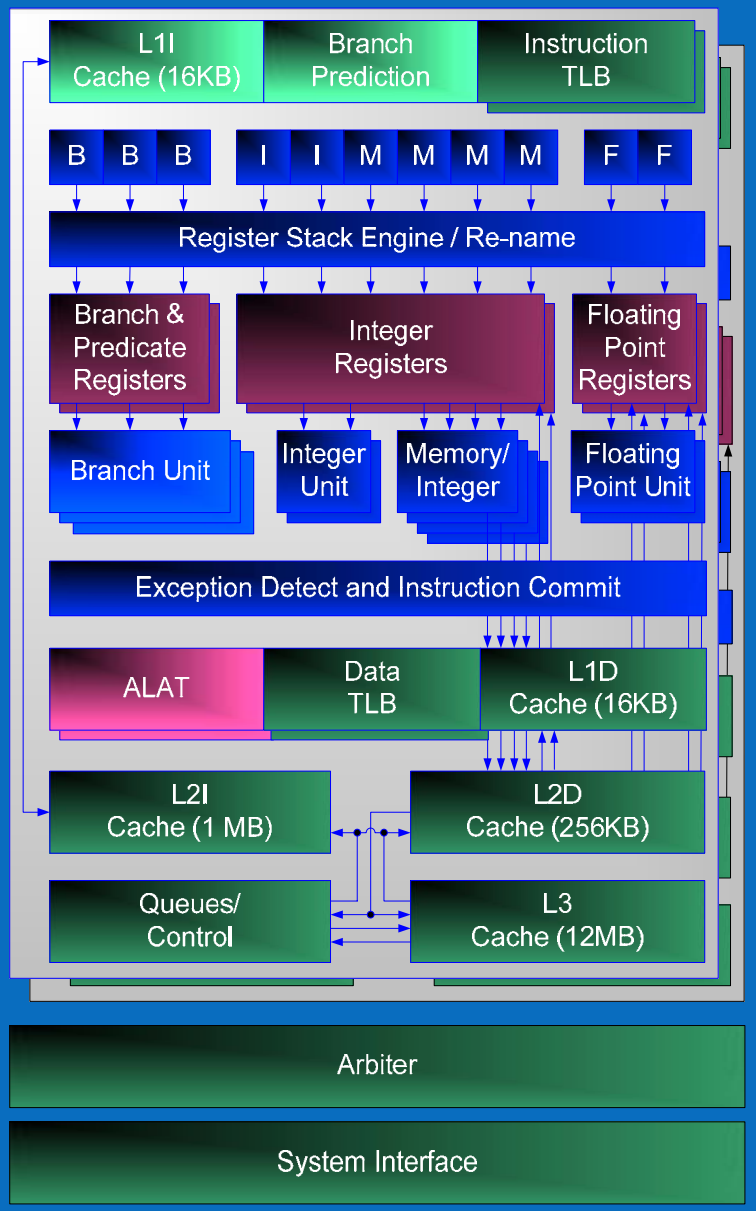
- Switch on event multi-threading
- Only 1 thread owns core resources at any time
- A pipeline flush cancels all in-flight unretired operations from a previous thread between switches
- 15 cycle cost for a thread switch

Memory hierarchy

- Simultaneous multi-threading
- Each thread competes for resources and is given equal access
- Competitively shared resources
  - I and D TLBs are tagged with thread identifier
  - Up to 32 TRs and at least 8 TCs for each thread
  - I and D Caches are tagged with thread identifier
  - Thread tracked only for register returns, ordering, and PMU
  - Branch prediction

Dedicated resources

- Architecture state: AR, BR, CR, FR, GR, PR, ...
- Micro-architecture structures such as ALAT, RSB, PIC, and PFM stack



Digital Enterprise Group

intel

Dynamic Thread Switching

Optimal: Determine when stalled for long latency operations

Practical: Speculate that a long latency event will stall execution and give software explicit switch control (`hint@pause`)

Switch Events

- L3 miss/return or UC accesses
  - Demand I or D but not prefetch I or D
  - HPW accesses that miss
- Time outs ensure fairness
- ALAT invalidation
- Low power HALT request

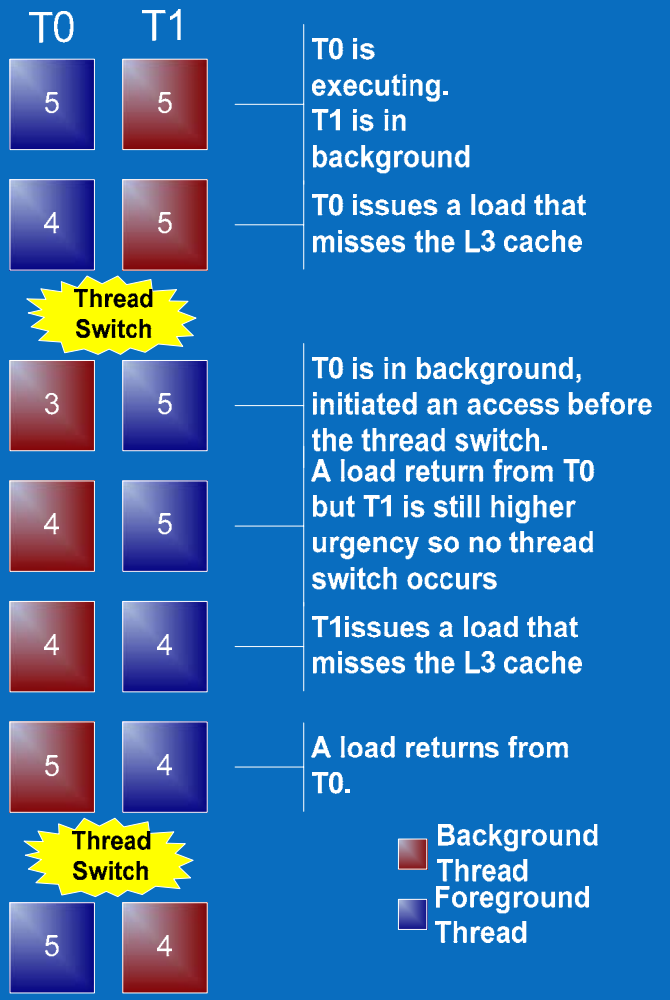
Hysteresis

- Urgency indicates a thread's ability to execute
- Compare urgency at miss and return events
- Latency from miss event to switch allows clustering
- Maximum switch thresholds ensure forward progress
- Switch may be delayed to ensure commit of pending state
  - i.e. long latency register writes, but NOT outstanding loads

Urgency Deep Dive

Each thread has an “urgency” counter that is compared at events and the thread with the greatest urgency executes

- Saturates at 0 and 5
  - L3 Miss event decrements urgency if <6 and >0
  - FSB/L3 data return increments urgency if <5
- Set to 5 at reset and >5 switch event
  - 6 = external interrupt occurred for background thread
  - 7 = thread switched out due to time slice expiration



Optimizing semaphores with the ALAT switch event

- hint @pause in combination with the ALAT switch event can be used to optimize semaphore performance under HT
- How it works:
  - The ALAT is coherent, such that a store by another processor in the coherent domain will result in an invalidation of any matching entry in the ALAT. A hint @pause instruction places the current thread in the background and enables any ALAT invalidation to bring the background thread to the foreground. If it is desirable to avoid bringing the waiting thread to the foreground as a result of invalidations of other (possibly stale) ALAT entries, the ALAT may be invalidated with an invala prior to the allocation of the ALAT entry for the memory location of interest.
  - Example code snippets on following slides

Spin Lock Code Example with ld.a + ld.c

```
spin_lock:
        invala                      // invala is optional and used to purge the ALAT
        mov     ar.ccv = 0          // so that only future address allocations will
        mov     r2 = 1              // initiate a switch event.
        ld8.a   r1 = [lock]         // Allocated the ALAT before entering loop
        ;;                          //
spin:
        ld8.c.nc r1 = [lock]        // ld.c.nc to check ALAT and allocate if miss
                                    // only issue load to memory hierarchy if miss
        cmp.eq  p1, p0 = r1, r2     // is lock busy?
        ;;
(p1)    hint    @pause              // yes, sleep until ALAT switch event
(p1)    br.cond.spnt spin           // yes, branch back.  Note, the branch
        ;;                          // retires before the thread switch
        cmpxchg8.acq     r1 = [lock], r2, ar.ccv
        ;;
        cmp.eq  p1, p0 = r1, r2
(p1)    br.cond.spnt spin
        ;;
cs_begin:
        …
cs_end:
        st8.rel  [lock] = r0
```

Spin Lock Code Example with ld.c

```
spin_lock:
        invala.e r1                 // need to purge ALAT for r1 since r1 is used
        mov      ar.ccv = 0         // in a ld.c below
        mov      r2 = 1
        ;;
spin:
        ld8.c.nc r1 = [lock]        // read variable and allocate this address
                                    // into the ALAT.  A stop bit here is optional
        cmp.eq   p1, p0 = r1, r2    // is lock busy?
        ;;
(p1)    hint@pause                  // yes, sleep until ALAT switch event
(p1)    br.cond.spnt spin           // yes, branch back.  Note, the branch
        ;;                          // retires before the thread switch
        cmpxchg8.acq      r1 = [lock], r2, ar.ccv
        ;;
        cmp.eq   p1, p0 = r1, r2
(p1)    br.cond.spnt spin
        ;;
cs_begin:
        …
cs_end:
        st8.rel  [lock] = r0        // free the lock
```


Notes/Caveats on the ALAT Switch Event

- The hint@pause should only be used when trying to acquire a semaphore – not when releasing the semaphore.
- The hint@pause will not switch until the issue group retires; it can be in the same issue group as a branch and both retire.
- When using ld.c as a replacement operation, please ensure the register is not in the ALAT with an invala.e.
- The invala/invala.e, if used, should scheduled no sooner than the issue group prior to the ALAT allocation and the hint@pause.
- The ALAT does not contain the full physical address, as such there may be false invalidations.
- A minimum of 15 cycles separate the switch event and retirement of the first issue group of the awakened thread. Most thread switch events, including the ALAT invalidation wake up event, may be delayed and increase this minimum.
- This thread switch event does not imply or enforce any priority between logical processors (threads, sockets, nodes, …); therefore, the semaphore may not be obtained directly after the thread comes to the foreground – races and delays may allow for another logical processor to obtain the semaphore.

Controlling Thread Switching Behavior

- HT personalities allow software to exert some control
  - Performance
  - Fairness
  - Priority
- Selectable via low latency PAL call, PAL_SET_HW_POLICY
  - PAL call adjusts the various switch events to affect thread switch behavior

Idle Loop Indication

- Software should indicate idleness via hint @pause
  - Will still get periodic wakeups due to thread timeout
- Use PAL_HALT_LIGHT PAL call to quiesce
  - Allows bias to running thread
  - Quiesced thread awakens only on interrupt

Hyper-Threading Implications

All instruction prefetching prefetches are canceled at thread switch

Branch prediction (L1I BR, PHT, L2B) are shared between threads

- L2B incorporates thread in access hashing algorithms

No high water limits on competitively shared resources except TLBs

- Each thread gets a minimum of 16 entries L2D TLB entries
- Each thread gets a minimum of 1 L2I TLB entry

TLB behavior

- The thread identifier is treated as an additional bit of VA
- L1I TLB allows PA alias between threads, but others prohibit PA alias

L1D behavior

- st.rel/semaphores on T0 may invalidate L1D line/request
  - An outstanding L1D fill to st.rel/semaphore line for T1
  - The st.rel/semaphore hit line brought in by T1
  - L1D fill buffers may also be invalided

Dual-Core Itanium® 2 9000 PMU

12 performance counters per thread

- 48 bits of resolution
- 200+ events can be counted

Where possible, the events are thread and core aware


intel
Leap ahead™


intel

References

- “Montecito: A Dual-Core, Dual-Threaded Itanium Processor”, *Cameron McNairy and Rohit Bhatia*, IEEE Micro Magazine, March/April 2005.
- Dual Core Update to the Intel Itanium 2 Processor Reference Manual, Document Number 308065-001